Core Java: JDBC I

Persistent Interactive | Persistent University

What is a Database?

- A repository of data
- Has control and management routines to hold and manage data
- A database is organised into tables.
- Every table has a name and is organised into columns and rows
- Each column has a name and a data type
- Each row represents a record in the database

Objectives :

- At the end of this module, you will be able to understand :
  - Database connectivity
  - Executing SQL statements (DDL, DML, DCL)
  - Use of ResultSet

What is SQL?

- Structured Query Language is a standard way to represent database commands called “queries”
- They can be typed directly into any database engine that supports SQL for example Oracle, DB2 etc.
- There are basically 5 types of SQL commands
  - DDL – Create, Alter, Drop, Truncate
  - DML – Insert, Update, Delete
  - TCL – Commit, Rollback, Savepoint
  - DCL – Grant, Revoke
  - DQL - Select

What is the need for JDBC?

- All databases support SQL (ANSI SQL)
- Different database vendors have introduced their proprietary SQL constructs
- Different database vendors have introduced Application Programming Interfaces for accessing data stored in their respective databases
- Languages such as C++ can directly access these proprietary APIs
- If the database changes, all data access logic has to be entirely re-written
- A need was felt to access data from different databases in a consistent and reliable way

What is JDBC?

- It is not an acronym, but is called Java Database Connectivity
- It is a vendor independent API drafted by Sun to access data from different databases in a consistent and reliable way
- JDBC provides an API by hiding the vendor specific API by introducing the concept of a JDBC driver between the application and the database API
- Hence, JDBC requires a vendor specific driver
- The JDBC driver converts the JDBC API calls from the Java application to the vendor specific API calls

JDBC

- If the database changes, the application can be configured to run with the new database by making very few changes in the code

- The database access in the application must be carefully designed to permit the almost transparent migration to the new database

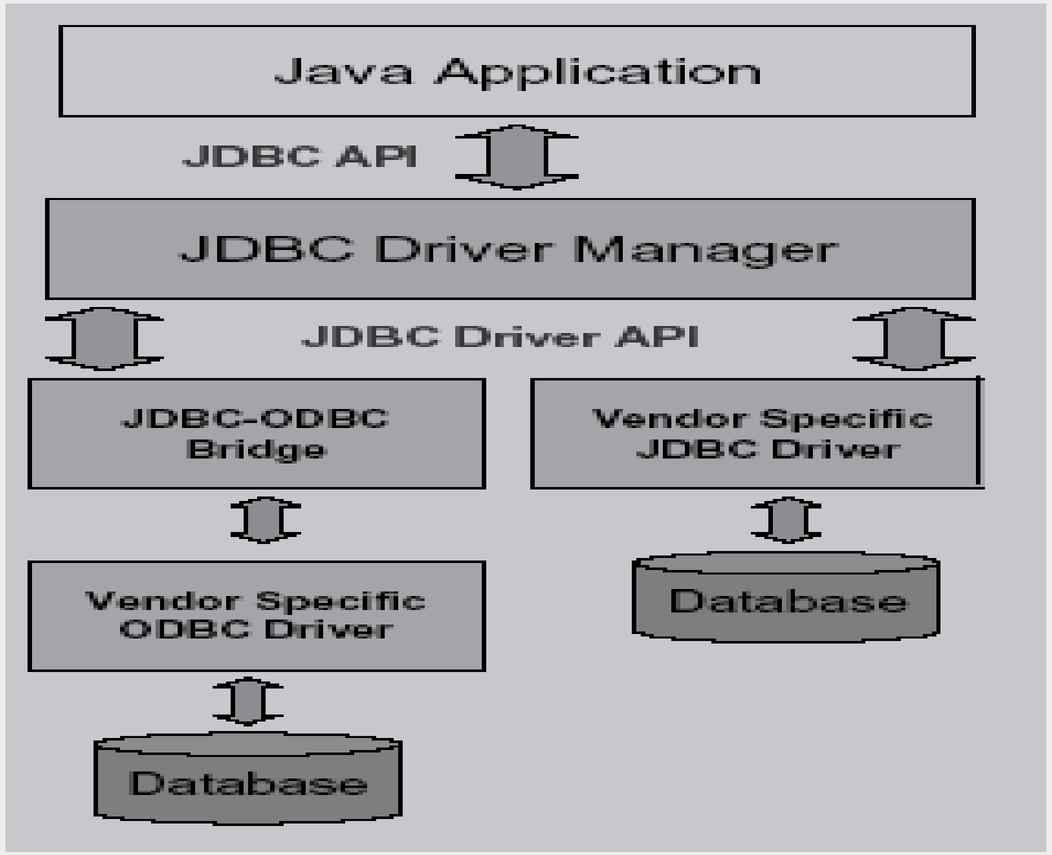
- JDBC requires the database vendors to furnish runtime implementation of its interfaces

Main goals of JDBC

- JDBC should be an SQL level API
- JDBC should capitalize on the experience of the existing database APIs
- JDBC should provide a simple programming interface

JDBC Architecture

- The JDBC architecture mainly consists of two parts
  - JDBC API, a purely Java based API
  - JDBC Driver Manager



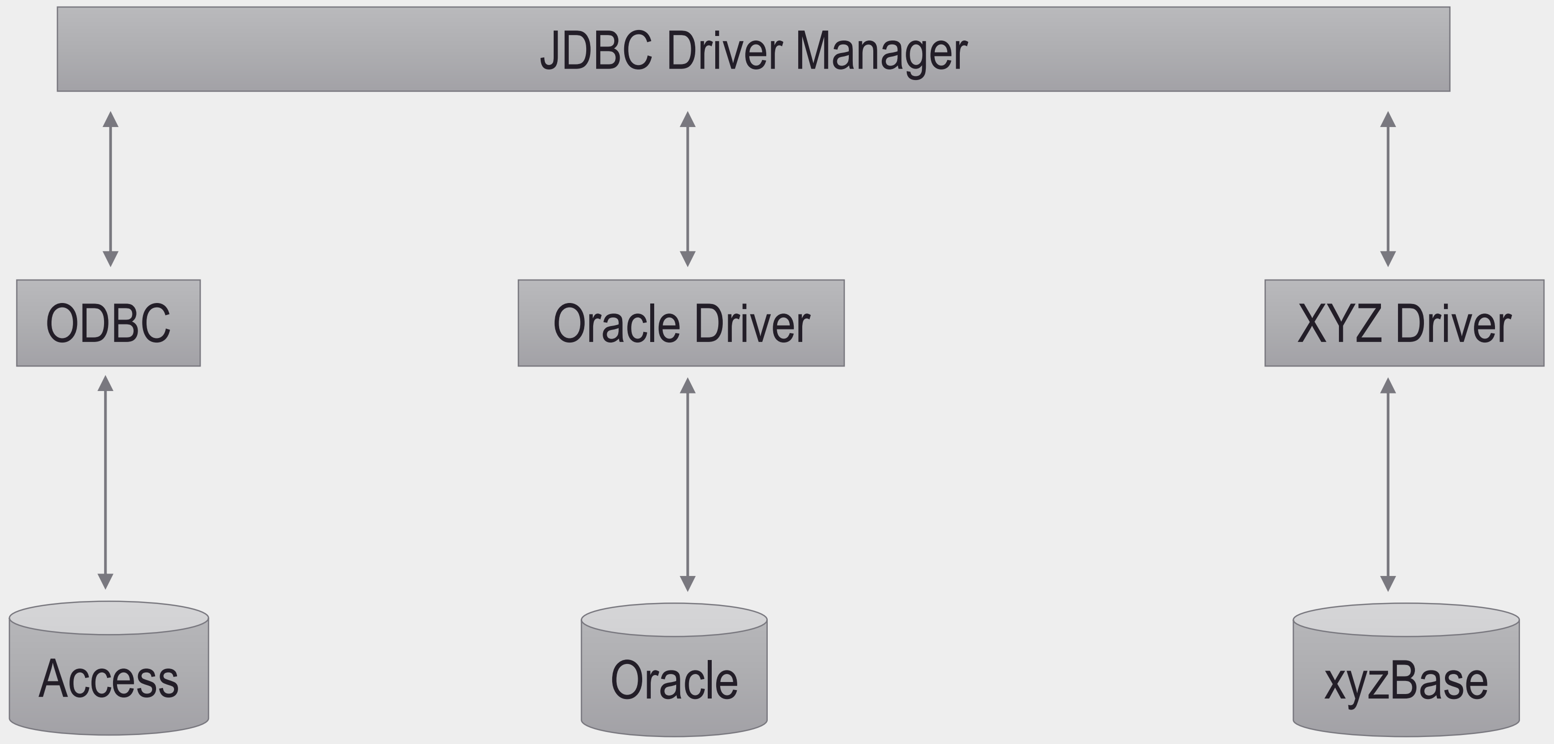
The Driver Manager

- It is possible that an application may need to interact with multiple databases created by different vendors

- The JDBC Driver Manager provides the ability to communicate with multiple databases and keep track of which driver is needed for which database

- Even if you need to interact with one database, you need to do it via the driver manager

The Driver Manager



Persistent

Basic steps in using JDBC

- Load the driver
- Define the connection URL
- Establish the database connection
- Create a statement object
- Execute query
- Process results
- Close database connection

Loading the JDBC Driver

- Manually register using Class.forName()

- A static block in the class automatically creates the instance and registers it with the driver manager

```
  try {
      Class.forName ("com.mysql.jdbc.Driver");
      }
catch (ClassNotFoundException ex)
   {
  System.out.println ("Error while loading driver class" +
  ex);
    }
```

Define the connection URL

- This step basically specifies the location of the database server
- URLs are used to specify the location of the database server
- The URLs must conform to the jdbc: protocol
- The URL must define the server host, the port, and the database name
- The exact format of the JDBC URL is defined in the JDBC driver's documentation

```
DriverManager.getConnection("jdbc:mysql://localhost:3306/emp", "root", "root");
```

Establishing the connection

- Connection object represent a DB connection
- This step creates the actual network connection with the database server. You need to pass the URL, user name, and password to the DriverManager’s getConnection () method
- getConnection () throws SQLException

```
try{
    Connection con =
        DriverManager.getConnection("jdbc:mysql://loc
alhost:3306/emp", "root", "root");
} catch (SQLException e) {
        e.printStackTrace();
}
```

Creating a Statement

- Statement object is used to send queries to the database
- It can be retrieved from the Connection object
- The different types of statements will be discussed later

```
//Assume that Connection object "conn" has already
    been created
  try {
            Statement stmt =
    conn.createStatement ();
    }
  catch (SQLException ex) {
  System.out.println ("Exception while retrieving meta-
    data");
                }
```

Execute a query

- Once you have a statement object, you can start executing queries

```
try {
        //Assume that Connection object “conn” has already been created
        //Assume that Statement object “stmt” has already been created
        String sqlQuery = “SELECT FIRSTNAME, LASTNAME, EMPID FROM EMPLOYEE”;
        ResultSet rs = stmt.executeQuery (sqlQuery);
}
catch (SQLException ex) {
        System.out.println (“Exception while retrieving meta-data”);
}
```

Closing the connection

- Remove the connection between the client and the database server
- To explicitly close the connection, use the close() method

```
try {
        conn.close ();
}
catch (SQLException sqle) {
        System.out.println ("Error occurred while
closing the connection");
}
```

Using Statement

- The Statement object is used to execute SQL queries against the database
- There are 3 types of Statament objects
  - Statement : For executing simple SQL statements
  - PreparedStatement : For executing pre-compiled SQL statements
  - CallableStatament : For executing database stored procedures

Using Statement methods

- **executeQuery ()**
  - Executes SQL query and returns data in a table (ResultSet)
  - The resulting table may be empty, but never null
  - ResultSet rs = statement.executeQuery (“SELECT * FROM EMPLOYEE”);

Using Statement methods

- **executeUpdate ()**
  - Used to execute INSERT, UPDATE, and DELETE SQL statements
  - It returns an int that represents the number of rows that were affected in the table
  - Supports Data Definition Language (DDL) statements
  - CREATE TABLE, DROP TABLE, ALTER TABLE
  - int nRows = statement.executeUpdate (“DELETE FROM EMPLOYEE WHERE EMPID = 22”);

Using Statement methods

- **execute ()**
  - Generic method for executing stored procedures and prepared statements
  - Rarely used for returning multiple result sets
  - The statement execution may or may not return a ResultSet
  - If statement.getResultSet returns true, two or more result sets were produced

Process results : Use of ResultSet

- The results of query execution are stored in a ResultSet object
- The standard practice is to process one row in the ResultSet at a time
- The ResultSet.next () method is used for such processing
- ResultSet supports getXXX () methods to retrieve column values. E.g. getString (), getInt ()
- getXXX () methods can be used with either column index or column name
- First column in a ResultSet has index 1, not 0

Process results : Use of ResultSet

Assume Connection object “conn” has been created

Assume Statement object “stmt” has been created

Assume ResultSet object “resultSet” has been obtained

```
try {
        while (resultSet.next ()) {
        String firstName = resultSet.getString (1);
        //getString (“FIRSTNAME”)
        String lastName = resultSet.getString (2);
        //getString (“LASTNAME”)
        int empID = resultSet.getInt (3);
         //getString (“EMPID”)
                }
      }
        catch (SQLException sqle) {
        System.out.println (“Error processing records”);
        }
```

Summary : Session

With this we have come to an end of our session, where we discussed about ….

- Basic concepts of JDBC
- Database connectivity
- Executing SQL statements
- Usage of ResultSet

Appendix

References

Thank you

Reference Material : Websites & Blogs

- http://www.tutorialspoint.com/jdbc/
- http://www.tutorialspoint.com/jdbc/jdbc-sample-code.htm
- http://www.javatpoint.com/java-jdbc

Reference Material : Books

- **Head First Java**
  - By: Kathy Sierra, Bert Bates
  - Publisher: O’Reilly Media, Inc.
- **Java Complete Reference**
  - By Herbert Schildt

Persistent

Thank you!

Persistent Interactive | Persistent University